Midterm II Review

AW

Lecture Overview

1. MT II Composition
2. Convolution with a given filter
3. Layer dimensionality in a convolutional system
4. Vanishing/Exploding Gradient
5. T/F/MC Questions
6. Qualitative graduate question

Questions and Grading

Composition:

- Undergrad/grad section – very similar to HWs
  - 3 open problem questions
  - T/F-MC section-4 questions
- Grad only section – not in HWs but in lectures
  - 1 open problem

Grading:

- Grades given in [ ] for undergrad, multiplier fraction for grad students in{ }
- Distribution of grades over questions: 20,20,40 MC/TF 5 each – total 100, max earned 70
- One question has partial credit
  - Updates for the network forward
  - Backward propagation

Lecture Overview

1. MT II Composition
2. Convolution with a given filter
3. Layer dimensionality in a convolutional system
4. Vanishing/Exploding Gradient
5. T/F/MC Questions
6. Qualitative graduate question

Compute Convolution Transformation

Compute the convolution of the input volume matrix with the horizontal edge detection filter. Use a stride of 1 without padding.

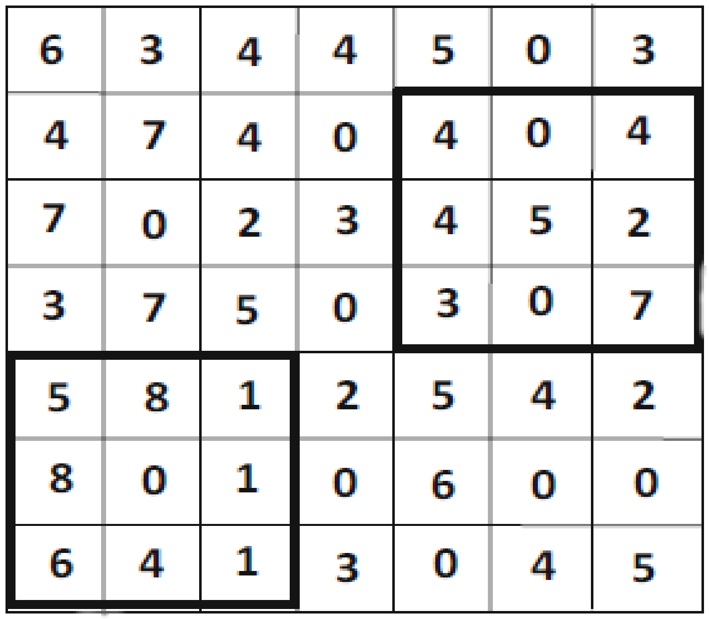
6	3	4	4	5	0	3
4	7	4	0	4	0	4
7	0	2	3	4	5	2
3	7	5	0	3	0	7
5	8	1	2	5	4	2
8	0	1	0	6	0	0
6	4	1	3	0	4	5

INPUT

1	1	1
0	0	0
-1	-1	-1

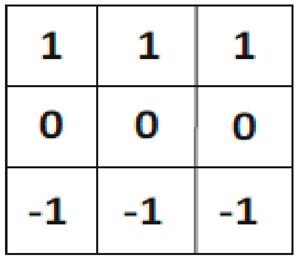
HORIZONTAL EDGE DETECTING FILTER

Solution: Compute Convolution Transformation



6	3	4	4	5	0	3
4	7	4	0	4	0	4
7	0	2	3	4	5	2
3	7	5	0	3	0	7
5	8	1	2	5	4	2
8	0	1	0	6	0	0
6	4	1	3	0	4	5

INPUT



1	1	1
0	0	0
-1	-1	-1

HORIZONTAL EDGE DETECTING FILTER

The final result of the convolution is a 5×5 output volume with the following entries:

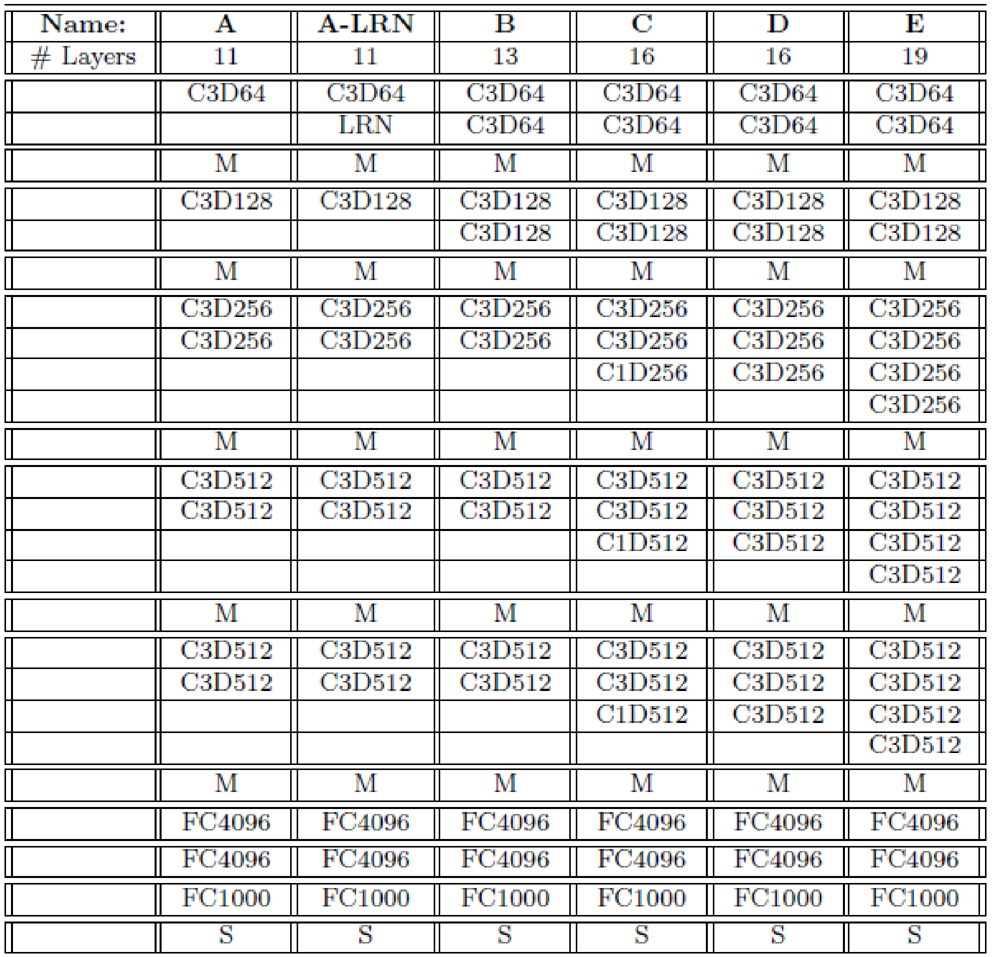
4	6	4	−3	−3
0	−1	0	1	−2
−5	−6	1	−1	0
6	11	1	−3	4
3	3	4	4	2

Lecture Overview

1. MT II Composition
2. Convolution with a given filter
3. **Layer dimensionality in CNN**
4. Vanishing/Exploding Gradient
5. T/F/MC Questions
6. Qualitative graduate question

Dimensionality of VGG 16

Work out the sizes of the spatial convolution layers for each of the columns of the table shown in the text on VGGNet (below). In each case, we start with an input image volume of $224 \times 224 \times 3$.



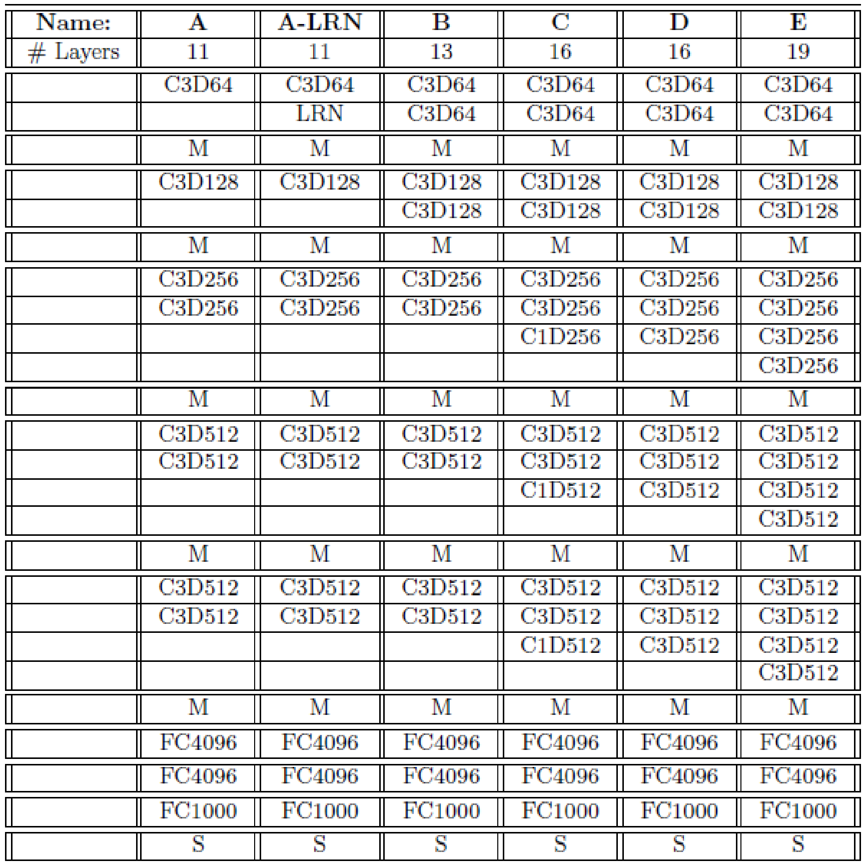
Name:	A	A-LRN	B	C	D	E
# Layers	11	11	13	16	16	19
	C3D64	C3D64	C3D64	C3D64	C3D64	C3D64
		LRN	C3D64	C3D64	C3D64	C3D64
	M	M	M	M	M	M
	C3D128	C3D128	C3D128	C3D128	C3D128	C3D128
			C3D128	C3D128	C3D128	C3D128
	M	M	M	M	M	M
	C3D256	C3D256	C3D256	C3D256	C3D256	C3D256
	C3D256	C3D256	C3D256	C3D256	C3D256	C3D256
				C1D256	C3D256	C3D256
						C3D256
	M	M	M	M	M	M
	C3D512	C3D512	C3D512	C3D512	C3D512	C3D512
	C3D512	C3D512	C3D512	C3D512	C3D512	C3D512
				C1D512	C3D512	C3D512
						C3D512
	M	M	M	M	M	M
	C3D512	C3D512	C3D512	C3D512	C3D512	C3D512
	C3D512	C3D512	C3D512	C3D512	C3D512	C3D512
				C1D512	C3D512	C3D512
						C3D512
	M	M	M	M	M	M
	FC4096	FC4096	FC4096	FC4096	FC4096	FC4096
	FC4096	FC4096	FC4096	FC4096	FC4096	FC4096
	FC1000	FC1000	FC1000	FC1000	FC1000	FC1000
	S	S	S	S	S	S

LRN – Layer: Local Response Normalization

- LRN is a non-trainable layer that square-normalizes the pixel values in a feature map within a local neighborhood
- This layer is useful when we are dealing with ReLU neurons and linear neurons.
  - ReLU/linear neurons have unbounded activations and we need LRN to normalize that.
  - It works within one filter and does not change the dimensionality of the output

Solution: Dimensionality of VGG 16

- Since convolutional volumes are maintained by the use of 3×3 filters with a padding of 1, the spatial sizes remain fixed in convolutional layers and depth is given by the number of filters. The max-pooling for VGG16 is $max(i, i+1)$ in both dimensions, so it reduces the each dimension spatial footprint by a factor of 2.
- So sizes are same for all columns
- Call a set of convolutional layers between max-pooling layers 'block'



Name:	A	A-LRN	B	C	D	E
# Layers	11	11	13	16	16	19
	C3D64	C3D64	C3D64	C3D64	C3D64	C3D64
		LRN	C3D64	C3D64	C3D64	C3D64
	M	M	M	M	M	M
	C3D128	C3D128	C3D128	C3D128	C3D128	C3D128
			C3D128	C3D128	C3D128	C3D128
	M	M	M	M	M	M
	C3D256	C3D256	C3D256	C3D256	C3D256	C3D256
	C3D256	C3D256	C3D256	C3D256	C3D256	C3D256
				C1D256	C3D256	C3D256
						C3D256
	M	M	M	M	M	M
	C3D512	C3D512	C3D512	C3D512	C3D512	C3D512
	C3D512	C3D512	C3D512	C3D512	C3D512	C3D512
				C1D512	C3D512	C3D512
						C3D512
	M	M	M	M	M	M
	C3D512	C3D512	C3D512	C3D512	C3D512	C3D512
	C3D512	C3D512	C3D512	C3D512	C3D512	C3D512
				C1D512	C3D512	C3D512
						C3D512
	M	M	M	M	M	M
	FC4096	FC4096	FC4096	FC4096	FC4096	FC4096
	FC4096	FC4096	FC4096	FC4096	FC4096	FC4096
	FC1000	FC1000	FC1000	FC1000	FC1000	FC1000
	S	S	S	S	S	S

- Since input image is $224 \times 224 \ \times 3$ (block, size) numbers are: $1\!: 3ch \times (224^2, 64); 2\!: 3ch \times (112^2, 128), 3\!: 3ch \times (56^2, 256),$ $4\!: 3ch \times (28^2, 512), 5\!: 3ch\!:\!\times (14^2, 512), f\!: (7^2, 512),$ $dfc1\!: 1 \times 4096, dfc2\!: 1 \times 4096,\ dfc3\!: 1 \times 4096$

Lecture Overview

1. MT II Composition
2. Convolution with a given filter
3. Layer dimensionality in CNN
4. **Vanishing/Exploding Gradient**
5. T/F/MC Questions
6. Qualitative graduate question

RNN with Fixed hh Matrix

Consider a recurrent network in which the hidden states have a dimensionality of 2. Every entry of the 2×2 matrix W_{hh} of transformations between hidden states is 3.5. Furthermore, sigmoid activation is used between hidden states of different temporal layers. Would such a network be more prone to the vanishing or the exploding gradient problem?

Hint: use wolfram alpha for spectral radius to find correct result.

Solution: RNN with Fixed hh Matrix

Matrix $M = \begin{pmatrix} 3.5 & 3.5 \\ 3.5 & 3.5 \end{pmatrix} = 3.5 \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$ has at most 2 distinct eigenvectors. Any vector of the form $\vec{a}_1 = \begin{pmatrix} a \\ a \end{pmatrix}$ since $M\vec{a}_1 = 7\begin{pmatrix} a \\ a \end{pmatrix}$ and another eigenvector $\vec{a}_2 = \begin{pmatrix} -a \\ a \end{pmatrix}$ since $M\vec{a}_2 = 0\begin{pmatrix} a \\ a \end{pmatrix}$, so the two eigenvectors are $\lambda_1 = 7, \lambda_2 = 0$. Spectral radius is 7.

On backpropagation we multiply by M^T to compute back gradient – it has the same spectral radius as M and M^T have same eigenvectors. So on a loss vector $\vec{l}$ we get on a path to compute $\nabla_{\vec{w}} L$ for input weight vector $\vec{w}$ we are getting the expression of the form $\underbrace{\Phi'\left(M^T\left(\Phi' \ldots \Phi'\left(M^T\vec{l}\right) \ldots\right)\right)}_{n-hidden\ neurons}$ where Φ' is derivative of sigmoid which may be as big as 0.5(1-0.5)=0.25. Thus

$$\underbrace{\Phi'\left(M^T\left(\Phi' \ldots \Phi'\left(M^T\vec{l}\right) \ldots\right)\right)}_{n-hidden\ neurons} \le (0.25M^T)^n\vec{l} = \left(\frac{7}{4}\right)^n \vec{l}$$ since values between 1.75 and 1 are not only possible but likely we are looking at exploding gradient problem

Lecture Overview

1. **MT II Composition**
2. **Convolution with a given filter**
3. **Layer dimensionality in CNN**
4. **Vanishing/Exploding Gradient**
5. **T/F/MC Questions**
6. **Qualitative graduate question**

T/F/MC Bagging/Dropout

- Dropout ANNs and bagging ANNs minimize the same functions
    - T or F?

T/F/MC Bagging/Dropout

Dropout ANNs and bagging ANNs minimize the same functions

- T [✓ F]

  dropout minimizes expectation of loss wrt to probability distribution of random dropout vector and network parameters, in bagging we minimize the average of expectation of losses of many networks, each wrt its own set of parameters (i.e. one for each network)

T/F/MC Pretrained CNNs

Training dense layers on very small training set of images for image classification with pretrained frozen CNNs usually gives ________ quality and requires _______ time than doing feature extraction with pretrained CNN and then feeding the extracted features to dense network

1. Better, more
2. Same, more
3. Lower, more
4. Lower, less
5. Same, less
6. Better, less

T/F/MC Pretrained CNNs

Training dense layers on very small training set of images for image classification with pretrained frozen CNNs usually gives ________ quality and requires _______ time than doing feature extraction with pretrained CNN and then feeding the extracted features to dense network

1. Better, more
2. Same, more
3. Worse, more
4. Worse, less
5. Same, less
6. Better, less

T/F/MC Pretrained CNNs

In recurrent network the total loss is sum of the losses over all time steps.

- T F

T/F/MC Pretrained CNNs

In recurrent network the total loss is sum of the losses over all time steps.

- [✓ T] F

  Since individual loss is categorical cross-entropy $\log p(y_t|x_1, \ldots, x_t)$ that depends on previous inputs and hidden states total loss must be computed as a sum of losses for all previous states $1, \ldots, t$.

T/F/MC Pretrained CNNs

Leaky Units

1. Have memory depth (number of state values remembered) set as a hyper parameter
2. Memory depth is partially controlled by a hyperparameter and partially learned
3. Memory depth is learned in training
4. Are gated units in which memory depth is gated by outputs of previous hidden units

T/F/MC Pretrained CNNs

Leaky Units

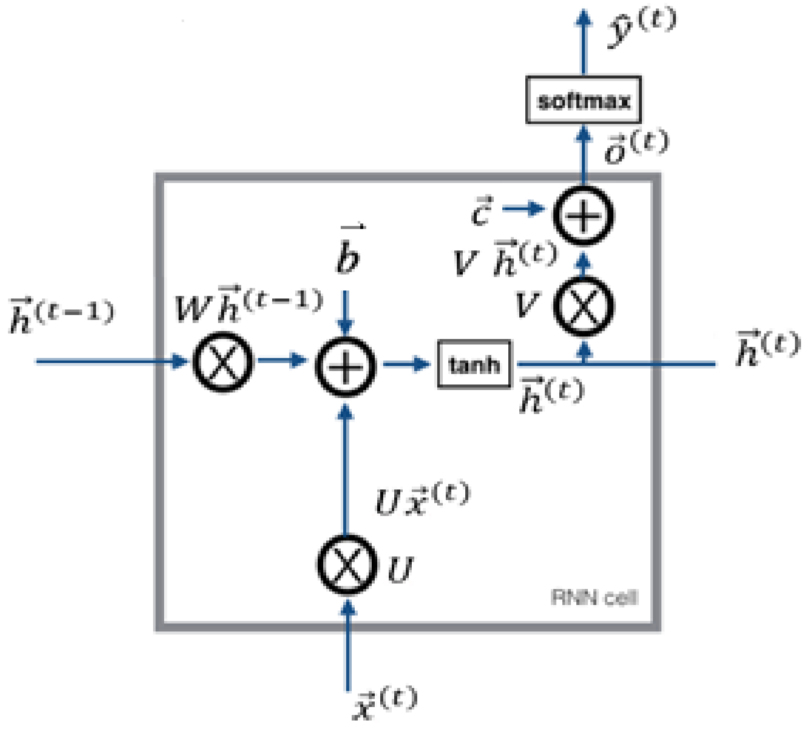
1. Have memory depth (number of state values remembered) set as a hyper parameter
2. Memory depth is partially controlled by a hyperparameter and partially learned
3. Memory depth is learned in training
4. Are gated units in which memory depth is gated by outputs of previous hidden units

Lecture Overview

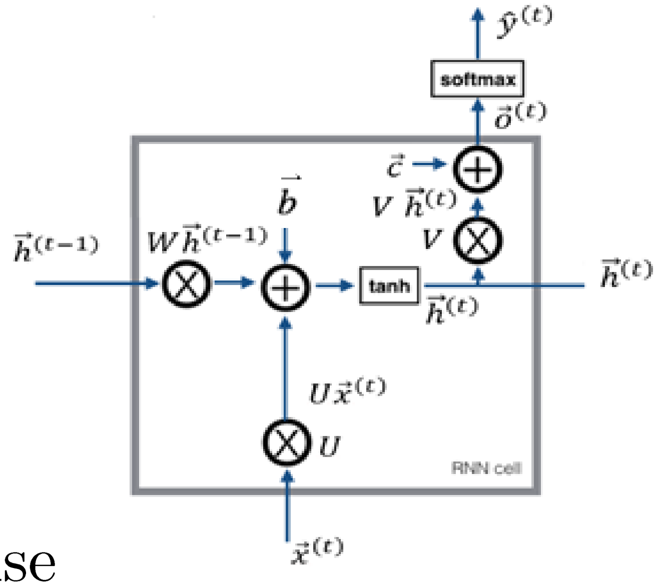
1. MT II Composition
2. Convolution with a given filter
3. Layer dimensionality in CNN
4. Vanishing/Exploding Gradient
5. T/F/MC Questions
6. **Qualitative graduate question**

RNN Cells Computational Graph

As can be seen below, RNN is composed of RNN cells similarly to LSTM or GRU. To see that consider RNN computation (i.e. formulas) as the definition of one cell computation. Show the computational graph of forward computation of one RNN cell.



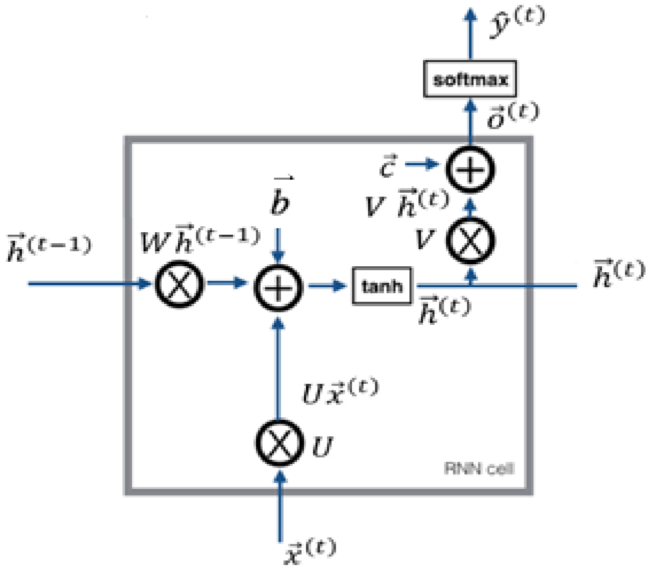
Solution: RNN Cells Computational Graph



The main caveat is that all weights are shared so the computational graph of the shell should reflect that (either by showing delay in folded cell or by showing unfolded cell structure). If we do not show the connections and just use W, U, V formulas without indices as if they are the same then the first backpropagation in computation will make all $W^{(t)}, U^{(t)}, V^{(t)}$ different after one backpropagation!

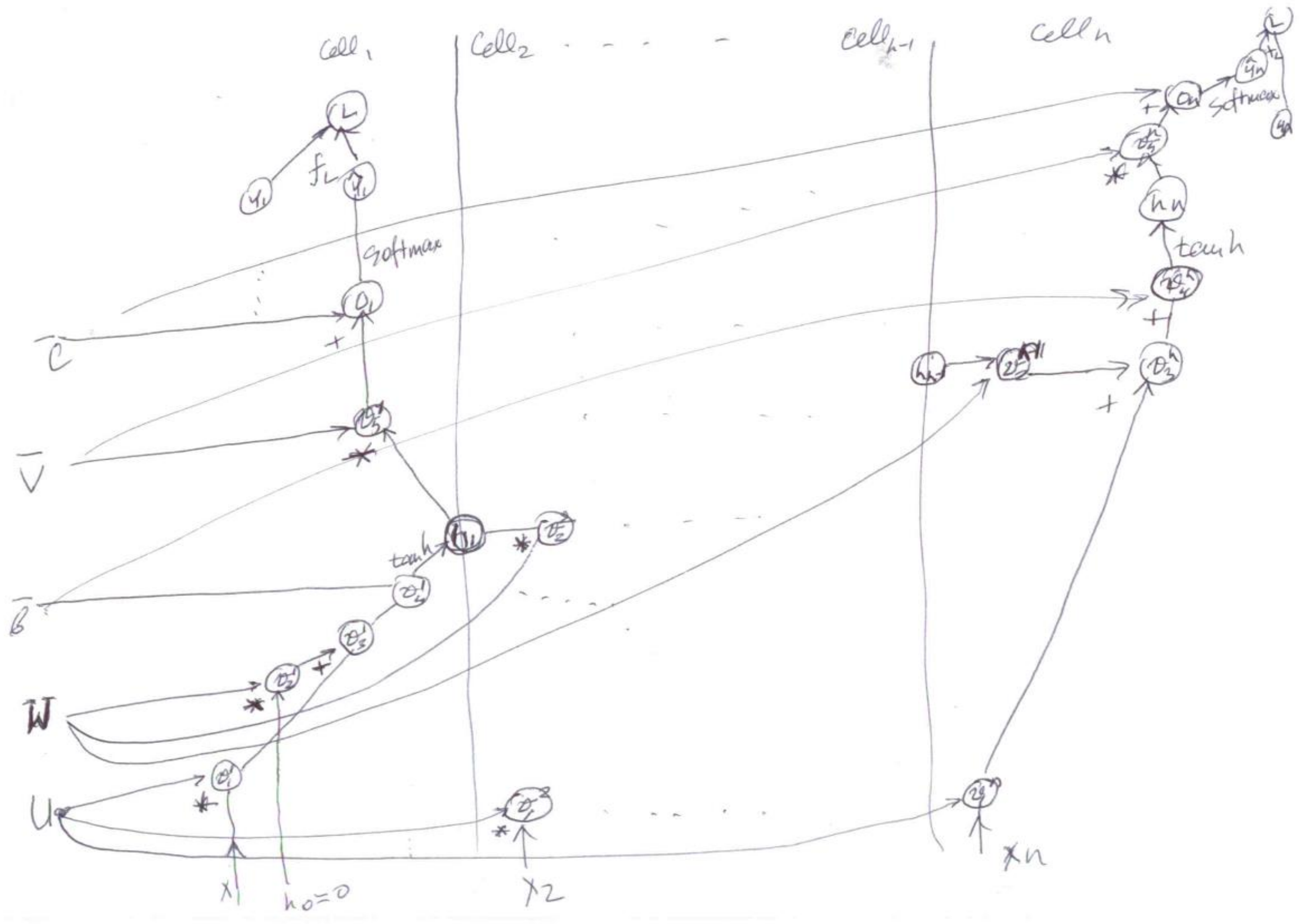
Solution: RNN Cells Computational Graph

The solution to sharing parameters is to make the gradient of shared parameters to have contribution of all edges that lead to this parameter being shared, so the total gradient is just the sum of gradients for all times



$\nabla_W L = \sum_{t=1}^{\tau} \nabla_{w^{(t)}} L_i$, which means that on the reverse graph there should be a path from each L_i to every parameter matrix $W^{(t)}, U^{(t)}, V^{(t)}$ and therefore these connections should be on forward computational graph too!

Solution: RNN Cells Computational Graph


Cell 1
Cell 2
Cell n-1
Cell n
L
f_L
softmax
tanh
+
*
C
V
B
W
U
x1
h0=0
x2
xn
tanh
softmax